OPEN ACCESS

University of BRISTOL

Amphlett, RW., & Bull, DR. (1996). Genetic algorithm based DSP multiprocessor scheduling. In *Unknown* (Vol. 2, pp. 253 - 256). Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/ISCAS.1996.540400

Peer reviewed version

Link to published version (if available):
10.1109/ISCAS.1996.540400

Link to publication record in Explore Bristol Research
PDF-document

University of Bristol - Explore Bristol Research
General rights

This document is made available in accordance with publisher policies. Please cite only the published version using the reference above. Full terms of use are available:
http://www.bristol.ac.uk/red/research-policy/pure/user-guides/ebr-terms/

GENETIC ALGORITHM BASED DSP MULTIPROCESSOR SCHEDULING

Robert W. Amphlett and David R. Bull
Centre for Communications Research,
University of Bristol, Bristol, UK.
R.W.Amphlett@bristol.ac.uk Dave.Bull@bristol.ac.uk

ABSTRACT

This paper presents recent work on the application of Genetic Algorithms to the NP-complete problem of multiprocessor scheduling for audio DSP algorithms. The Genetic Algorithm is used to schedule algorithms written in the form of data flow graphs onto specified multiprocessor arrays. A unique chromosome representation technique is described and a number of application-specific genetic operators are introduced. Comparisons of the performance of the Genetic Algorithm technique with heuristic scheduling techniques show that the choice of the most suitable technique varies with the structure and complexity of the scheduling problem. Finally, techniques for combining heuristic and Genetic Algorithm scheduling techniques are discussed.

1. INTRODUCTION

The use of large multiprocessor systems to solve real-time digital signal processing (DSP) problems has introduced the need for efficient task scheduling algorithms. For example, the processing requirements (>1000 MIPS, [1]) of multiple channel audio processing in studio mixing consoles far exceed the capabilities of single DSPs, therefore demanding the use of multiprocessor techniques. With the increasing complexity of DSP algorithms and the subsequent increase in the number of processing elements (PEs) being incorporated into processing arrays, the problem of efficiently scheduling a given algorithm onto a given multiprocessor network becomes non-trivial.

In recent years several techniques have been proposed as a solution to the solution of the NP-complete problem of multiprocessor mapping and scheduling [2], each with varying degrees of success [2-7]. Such methods include heuristic algorithms [2, 3], critical path techniques [4] and acyclic and cyclic graph methods [5]. An alternative, more recent approach, has involved the application of Genetic Algorithms (GAs) to this problem [6, 7].

This paper describes the application of GA-based optimisation methods to the problem of multiprocessor scheduling for audio signal processing programs. A suitable problem representation is described in section 2. Various application-specific genetic operators, designed to optimise the effectiveness of the GA search, are then introduced. Results showing the relative performance of the genetic operators are discussed in section 3 and the performance of the GA based scheduling technique is compared with the Greedy Algorithm heuristic in section 4. Finally, methods for obtaining near optimal scheduling performance by combining heuristic and GA based techniques are discussed.

2. GA BASED SCHEDULING

The principles of GAs were first developed by Holland [8] in 1975, and are well documented in many texts [9, 10]. GAs are based on the adaptive processes of natural systems which are essential for evolution, using direct analogies of natural behaviour such as 'populations' of 'chromosomes', 'reproduction', 'cross-breeding' and 'mutation'. They have been shown to be robust stochastic searching algorithms for a wide range of problems. The standard GA can be represented as shown in figure 1. The problem to be solved is represented as a set of parameters which can be encoded as a string (or 'chromosome') representing a potential solution to the problem. An objective (or 'fitness') function is then designed to evaluate how good a particular solution is at solving the problem.

```
BEGIN /* Genetic Algorithm */
    generate initial population
    compute fitness of each individual
    WHILE NOT finished DO
    /* produce new generation */
        FOR population_size/2 DO
        /* reproduction cycle */
            select 2 chromos. from old gen. for mating
            /* biased in favour of fitter individuals */
            combine the two chromos. to give two offspring
            /* crossover/mutation */
            compute fitness of offspring
            insert offspring into new generation
        END
        IF population has converged THEN
            finished = TRUE
    END
END
```

Fig. 1: Traditional Genetic Algorithm Structure

In order to apply Genetic Algorithms to the problem of multiprocessor scheduling, the DSP program is written in

0-7803-3073-0/96/$5.00 ©1996 IEEE

Authorized licensed use limited to: UNIVERSITY OF BRISTOL. Downloaded on February 9, 2009 at 10:06 from IEEE Xplore. Restrictions apply.

the form of a directed acyclic data graph [2]. It is then grouped into sets of tasks representing instruction blocks with high data-dependent edges. These tasks are organised in the form of a task graph (figure 2), which describes the node execution times, execution precedences and data transfers. The main GA maps the tasks onto a predefined multiprocessor array by assigning tasks to processors as ordered task lists (figure 3) while maintaining task precedences. The resulting schedule is determined from the ordered task list by taking into account each node's execution finishing time (figure 4).

Before the GA can begin processing, an initial population of legal schedules is produced by assigning tasks from each precedence level in the ordered task graph in turn (randomly) to the available processors. A special chromosome representation has been developed in order to fully represent legal solutions to the scheduling problem. Each chromosome is represented by two arrays. The first is a two-dimensional NxM array, where N is the number of processors and M is the maximum number of tasks that can be stored on each processor. This array stores the task numbers that make up the entire schedule, with a variable number of tasks allocated to each processor in their order of execution. The second array is one-dimensional with

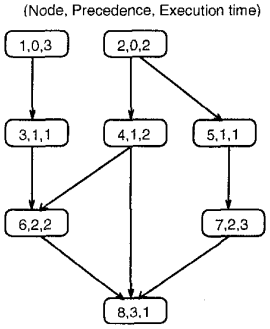


Figure 2: Example Task Graph

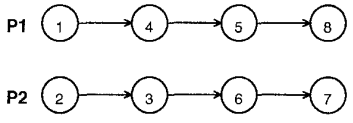


Figure 3: Ordered List (Two PEs)

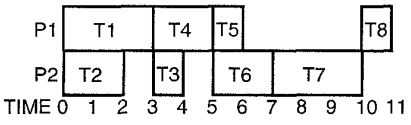


Figure 4: Example Schedule (Two PEs)

dimension N and stores the actual number of tasks allocated to each processor. Only legal schedules are allowed to be represented by the array structure. A legal schedule must satisfy the following criteria:

1. *All tasks are present at least once but once only.*
2. *The precedence relationships are maintained between tasks which are scheduled to the same processor.*

The GA proceeds by creating new subsequent populations of legal schedules while attempting to optimise a number of possible scheduling objectives. These include minimising the schedule finishing time, maximising processor utilisation and minimising costly inter-processor communication.

In order to maintain legal schedules and to ensure that the entire solution search landscape is covered by the GA, the chromosome structures described above are complemented by specific genetic operators. These are modified versions of the traditional binary crossover and mutation operators and include:

1. *One-Point Crossover*
2. *Task-Swap Mutation*
3. *Two-Point Crossover*
4. *Shuffle Mutation*
5. *N-Point Crossover*

All operations are limited to act on a precedence level basis. Crossover is limited to locations at the boundary between precedence levels, while mutation only acts upon tasks from individual precedence levels.

3. GENETIC OPERATOR RESULTS

Typical execution times on a Sparc 10 workstation for 100 generations ranged from approximately one minute for a 64 task schedule, to 1.2 hours for a 1000 task graph (typically 10,000 instructions). The simpler 64 task graph case is presented below.

All possible combinations of operators have been applied to the problem of scheduling a 64 task graph onto 4 processors. Figure 5 shows example results, comparing one-point crossover/task-swap mutation with n-point crossover/shuffle mutation. Each graph shows the best and the standard deviation of the excess minimum finishing time for the schedule found at each generation of the trial by the GA (i.e. the fitness is calculated as the amount by which the schedule length exceeds the critical path length, measured in program instruction cycles). All GAs are successful at reducing the finishing time of the schedule but some techniques perform better than their counterparts. In the examples shown, it can be seen that n-point crossover with shuffle mutation achieves a better excess

Authorized licensed use limited to: UNIVERSITY OF BRISTOL. Downloaded on February 9, 2009 at 10:06 from IEEE Xplore. Restrictions apply.

minimum finish time (47 instructions) than one-point crossover with task-swap mutation (92 instructions). The standard deviation shown in figure 5a rapidly approaches zero, indicating the possible premature convergence of the GA population. This could contribute to the poorer performance of the GA employing one-point crossover and task-swap mutation.

4. A COMPARISON BETWEEN THE GA AND A GREEDY ALGORITHM

In order to evaluate the comparative performance of the GA method, it was applied to a range of task graphs of varying complexity. The complexity of each graph is measured by comparing the absolute minimum finishing time of the schedule, using critical path analysis (assuming an unlimited number processors), with the 'mean load per processor' calculated as the sum of the execution times of all tasks divided by the number of processors. The last of these two measures acts as the limiting factor which determines the minimum finishing time possible for a given schedule. For example, four different 64-task graphs were scheduled for 100 generations. The results are presented graphically in figure 6, which shows the best, mean and worst schedule fitnesses (i.e. instruction cycles in excess of critical path schedule) for each GA generation.

In table 1, the minimum schedule finishing time results are compared with those from a greedy algorithm. The greedy algorithm is a commonly used approach which produces a single schedule solution by recursively assigning the largest task on each successive precedence level to the next available processor. Table 1 shows that, for simpler graphs, or conversely a larger number of available processors, simple heuristic algorithms, such as the greedy algorithm can be successful at finding optimal or near optimal solutions. However, for harder scheduling problems they are not as effective at reducing schedule finishing times. In such cases, the GA technique performs better, offering a reduction in finishing time of up to 85% for a 64 task / 8 processor problem.

A current area of investigation involves hybridisation of heuristic and GA techniques in order to utilise the best aspects from both. A simple example of this involves taking the solution from the greedy algorithm to a task graph and using it to 'seed' the initial population of the GA. Early results have found that this approach can produce better results than either technique in isolation. However, care must be exercised in order to prevent the fitness of the greedy solution dominating the rest of the GA population, preventing the formation of alternative minimum finishing time schedules.

5. CONCLUSIONS

The results presented here demonstrate that the performance of GA based multiprocessor scheduling compares favourably with existing heuristic methods. Modifications made to the GA operators, such as crossover and mutation, do not significantly affect the GAs performance provided that the legality of schedules is preserved. Finally, techniques for hybridising Genetic Algorithms with existing multiprocessor scheduling techniques have been introduced. Early results indicate that these will out-perform existing techniques for most applications.

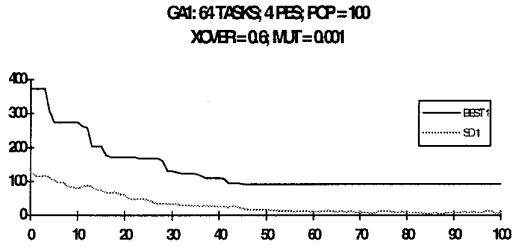
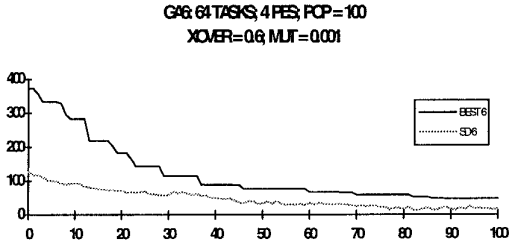


Figure 5 (a): 1-Point Crossover, Task Swap Mutation



(b): N-Point Crossover, Shuffle Mutation

Graph No.	Tasks	Processors	Critical Path Length	Mean Processor Load	GA Min. Finish Time Excess	Greedy Finish Time Excess
1	64	8	877	376	0	0
2	64	4	877	752	21	145
3	64	8	482	377	19	38
4	64	4	482	754	290	357

Table 1: GA - Greedy Algorithm Comparisons

Authorized licensed use limited to: UNIVERSITY OF BRISTOL. Downloaded on February 9, 2009 at 10:06 from IEEE Xplore. Restrictions apply.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of Sony Broadcast and Professional Europe and The Centre for Communications Research, University of Bristol.

REFERENCES

[1] Eastty, P., 'Digital Audio Processing on a Grand Scale', 81st AES Convention, Los Angeles, USA, Nov. 1986.

[2] Coffman, E.J., 'Computer and Job-Shop Scheduling Theory', John Wiley & Sons, 1976.

[3] Lo, V.M., 'Heuristic Algorithms for Task Assignment in Distributed Systems', IEEE Trans. Computers, Vol. 37, No. 11, Nov. 1988, pp. 1384-97.

[4] Kohler, W.H., 'A Preliminary Evaluation of the Critical Path Method for Scheduling Tasks on Multiprocessor Systems', IEEE Trans. Computers, Vol. C-24, Dec. 1975, pp. 1235-8.

[5] Ha, S. and Lee, E.A., 'Quasi-Static Scheduling for Multiprocessor DSP', 1991 IEEE Int. Symposium on Circuits and Systems, Vol. 1, Singapore, June 1991, pp. 352-5.

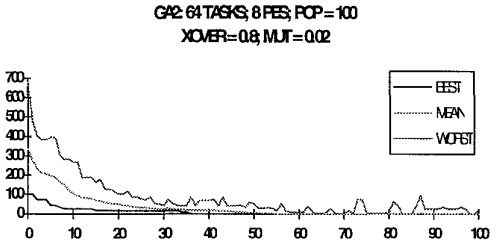
[6] Amphlett, R.W. and Bull, D.R., 'Multiprocessor Scheduling for High Quality Digital Audio', IEE Col. Multiprocessor DSP - Applications, Algorithms and Architectures, London, May 1995, pp. 2/1-2/8.

[7] Hou, E.S.H., Hong, R. and Ansari, N., 'Efficient Multiprocessor Scheduling Based on Genetic Algorithms', IECON '90, IEEE Industrial Electronics Soc., Pacific Gro. FL., USA, Vol. 2, Nov. 1990, pp. 1239-43.

[8] Holland, J.H., 'Adaptation in Natural and Artificial Systems', MIT Press, 1975.

[9] Goldberg, D.E., 'Genetic Algorithms in Search, Optimisation and Machine Learning', Addison-Wesley Publishing, 1989.

[10] Beasley, D., Bull, D.R. and Martin, R.R., 'An Overview of Genetic Algorithms', University Computing, 1993, Vol. 15, pp. 58-69, 170-181.



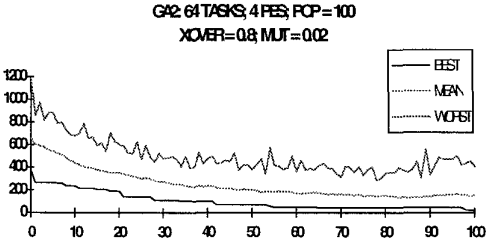
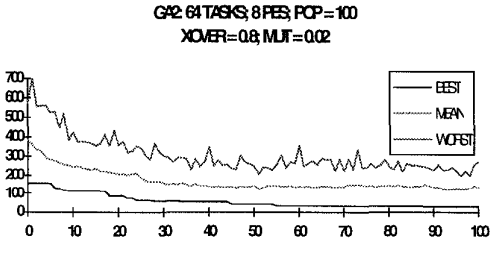
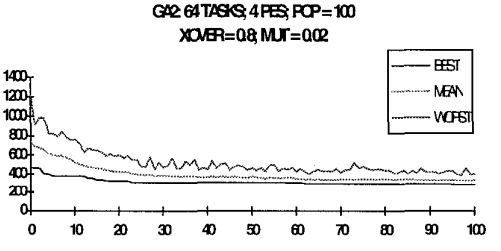


Figure 6 (a): 64 Tasks, 8 Processors; Mean Load 376 (b): 64 Tasks, 4 Processors; Mean Load 752
(100 Generations; Critical Path Length = 877 instruction cycles)





(c): 64 Tasks, 8 Processors; Mean Load 377 (d): 64 Tasks, 4 Processors; Mean Load 754
(100 Generations; Critical Path Length = 482 instruction cycles)

Authorized licensed use limited to: UNIVERSITY OF BRISTOL. Downloaded on February 9, 2009 at 10:06 from IEEE Xplore. Restrictions apply.